(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 December 2001 (06.12.2001)

PCT

(10) International Publication Number
WO 01/93019 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/IB01/00947

(22) International Filing Date: 30 May 2001 (30.05.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/584,448 31 May 2000 (31.05.2000) US

(71) Applicant: ADC TELECOMMUNICATIONS ISRAEL LTD. [IL/IL]; Ha'Sadnaot St. 10, 46728 Herzlia (IL).

(72) Inventor: RABINOVITCH, Shlomo; Greenspan 13/27, 75502 Rishon LeZion (IL).

(74) Agent: REINHOLD COHN AND PARTNERS; P.O. Box 4060, 61040 Tel Aviv (IL).

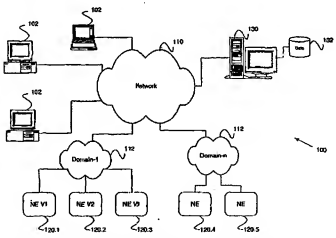
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: MANAGING COMPATIBILITY OF SOFTWARE IN DISTRIBUTED SYSTEMS



(57) Abstract: Systems and methods for managing versions of software components in a distributed system are disclosed. One aspect of the system is an application bank that stores various versions of applications that are designed to communicate with corresponding version of software running on a component element. Remote clients desiring to communicate with the component element determine the version of the application that corresponds to the version running on the component element. The remote client then downloads the appropriate version of the application (which may include multiple application programs) if it does not currently exist on remote client machine. A further aspect of the system is that the component element is a network element in a telecommunications system and the application is an element management application. In a still further aspect of the system, the element management application comprises a user interface component, an application component, and a communication component, all of which are separately versioned. A client desiring to execute the element management application downloads the appropriate version of each individual component if required.

WO 01/93019 A2

MANAGING COMPATIBILITY OF SOFTWARE IN DISTRIBUTED SYSTEMS

Field

The present invention relates generally to computer software systems, and more particularly to maintaining compatibility between differing versions of components of a multi-component system such as a distributed system.

Copyright Notice/Permission

Background

Software systems typically undergo many changes as the systems move through the software life cycle of development, test, release and maintenance. Even after systems have been released to customers (i.e. placed in the field) the software continues to evolve as corrections are applied and new features are added. As these corrections or features are added, new versions (also known as releases) of the software are generated and deployed to customers so that they can take advantage of the corrections or features. However, not all customers choose to install the new release. Reasons for this vary, for example some customers do not wish to pay for upgrades and corrections, other customers do not wish to upset a stable running system to gain features or corrections that provide no benefit to their particular environment.

The proliferation of versions can cause complications in the case of distributed

systems or systems with multiple constituent components. A distributed system is one in which the system functionality is distributed across several components. In this case, the various components must communicate with one another. Typically, a software application that communicates with another application is designed to communicate with a particular version of the application and perhaps older versions (e.g. if the applications are intended to be backward compatible). However, if newer applications are released, older applications may not be able to communicate with the newer applications. Furthermore, the set of features available in each version can change. Consider the case where a new version is released every six months. Further assume that each release puts approximately 100 systems into the field. In this case after three releases there are over 300 systems in the field, with each 100 systems comprising a different version. Thus, if one application needs to be able to communicate with all of the other released versions of the applications, the potential exists that it must support three different communications mechanisms. Moreover, the application will have to keep track of what features are available in the particular versions of the component software.

An example of the above-mentioned issues occurs in telecommunications networks. In such networks, management applications (referred to as Element Management Systems, or EMS) must be able to communicate with network elements (Nes). Each network element is typically running a software application at a particular version level, and the management software is also at a particular version. Thus there are potentially many different combinations of versions of management applications to network element applications. If a management application must communicate with all of the network elements, it must deal with every possible combination of network element software versions that it encounters.

A further complication occurs when the software must be tested. In this case, the application must typically be tested with all known versions of other components of the distributed system. For example assume that a software component of a network element has evolved over time into several versions 1, 2, 3, 4, and 5. Further assume

that an application, such as a management application that communicates with the component applications, has evolved over time into versions A, B, C, D, and E. accordingly. This problem is exemplified in Table 1 below, which represents a compatibility-testing map.

Evolution	Test No.	New Application version	Software version
NE v-I; EMS v-A.	1	v-A	v-1
NE v-2; EMS v-B.	1	v-B	v-1
	2	v-B	v-2
NE v-3; EMS v-C.	1	v-C	v-1
	2	v-C	v-2
	3	v-C	v-3
NE v-4; EMS v-D.	1	v-D	v-1
	2	v-D	v-2
	3	v-D	v-3
	4	v-D	v-4
NE v-5; EMS v-E.	1	v-E	v-1
	2	v-E	v-2
	3	v-E	v-3
	4	v-E	v-4
	5	v-E	v-5

Table 1.

As can be seen from the above table, the testing and compatibility issues increase as more versions exist. This leads to greatly increased test time and a corresponding increase in development costs. Moreover, there is increased risk that testing will overlook important feature compatibility.

Previous systems have attempted to deal with various aspects of the problem described above. One method of addressing the problem is to handle issues in the communication protocol used to communicate between the various system components.

In this method, two approaches have been used. The first is to design a protocol with a "wide" aspect. In this approach, the protocol is initially designed with spare fields, and spare message areas to allow room to add new fields and messages to the protocol. The protocol may be divided into several functional areas, with spare fields and messages allocated for each functional area. This approach attempts to predict where future development may take the protocol, and reserve room in the protocol to accommodate the evolution.

A second approach is to design a communications protocol with a message set that meets the system's current needs, with some prediction as to future needs. Later, as the system evolves, new message sets are added that implement new features and functionality. Compatibility with old message sets is maintained as long as deemed necessary, and when such compatibility is no longer required, the old message set is dropped.

In both approaches, an attempt is made to predict the direction of future evolution, and to define, or at least save room for providing additional capability. In each approach, the developer must keep in mind issues related to backwards compatibility and cross-compatibility, which become more complicated as more versions are created.

A further method used to address the compatibility issue is to require that all components of a distributed or multi-component system be upgraded to the latest version any time any component is upgraded to a new version. This method may be practical for relatively small systems, however it becomes unrealistic as the number of systems and components grows.

As a result, there is a need in the art for a system that can provide for the evolution of software elements in a distributed system that reduces the complexity and testing requirements when new versions of the software elements are introduced.

Summary

The above-mentioned shortcomings, disadvantages and problems are addressed

by the present invention, which will be understood by reading and studying the following specification.

In the various embodiments of the present invention, an application bank stores various versions of applications that are designed to communicate with corresponding versions of software running on a component element. Remote clients desiring to communicate with the component element determine the version of the application that corresponds to the version running on the component element. The remote client then downloads the appropriate version of the application if it does not currently exist on remote client machine.

In one embodiment, the component element is a network element in a telecommunications system and the application is an element management application.

In a further embodiment, the element management application comprises a user interface component, an application component, and a communication component, all of which are separately versioned. A client desiring to execute the element management application downloads the appropriate version of each individual component if required.

The present invention describes systems, clients, servers, methods, and computer-readable media of varying scope. In addition to the aspects and advantages of the present invention described in this summary, further aspects and advantages of the invention will become apparent by reference to the drawings and by reading the detailed description that follows.

Brief Description Of The Drawings

FIG. 1 is a block diagram of the hardware and operating environment in which different embodiments of the invention can be practiced;

FIG. 2 is a diagram providing further details on a computer system according to an embodiment of the invention;

FIG. 3 is a diagram illustrating providing an overview of software components according to an exemplary embodiment of the invention; and

FIG. 4 is a flow diagram illustrating a method of an exemplary embodiment of the invention.

Detailed Description

In the following detailed description of exemplary embodiments of the invention, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration specific exemplary embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that logical, mechanical, electrical and other changes may be made without departing from the scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense.

In the Figures, the same reference number is used throughout to refer to an identical component that appears in multiple Figures. Signals and connections may be referred to by the same reference number or label, and the actual meaning will be clear from its use in the context of the description.

The detailed description is divided into multiple sections. In the first section the hardware and operating environment of different embodiments of the invention is described. In the second section, the software environment of varying embodiments of the invention is described. In the final section, a conclusion is provided.

Hardware and Operating Environment

FIG. 1 provides a general description of an operating environment 100 in which embodiments of the invention may be practiced. Environment 100 includes remote workstations 102, and remote management server 130, all communicably coupled via network 110. In some embodiments of the invention, network 110 is includes one or more domain networks 112. Network elements 120 are communicable coupled to the one or more domain networks 112. Thus, as shown in FIG. 1, the remote workstations 102, the remote management server 130 and the various network elements 120 can all

communicate and transfer data between one another.

Remote workstations 102 comprise computer systems running applications that desire to communicate with software running on network elements 120. In one embodiment of the invention, remote workstations 102 are computer systems that run versions of the Microsoft Windows operating system, such as Windows 95, Windows 98, Windows NT or Windows 2000. In an alternative embodiment of the invention, remote workstations 102 run a variant of the UNIX operating system, such as Linux, HP-UX from Hewlett Packard Corp., or Solaris from Sun Microsystems, Inc. Embodiments of the invention are not limited to any particular operating system.

Network Elements 120 comprise computer systems that provide functions in a distributed computer system. In one embodiment of the invention, network elements 120 comprise components of a telecommunications network. In this embodiment, the components provide specialized functionality for the distributed system. As an example, the network element 120.1 can provide POTS (Plain Old Telephone Service) functionality, 120.2 can provide ISDN (Integrated Services Digital Network) functionality element, and 120.3 can provide T1 functionality. Other network elements can provide SDH (Synchronous Digital Hierarchy), ATM (Asynchronous Transfer Mode), ADSL (Asymmetric Digital Subscriber Line), TR-303/TR-08 and Leased Line functions. The invention is not limited to any particular network element function or combination of network element functions.

Remote server 130 is a conventional server system, as is known in the art. Remote server 130 comprises a computer system suitably configured to provide client systems access to resources resident on remote server 130, such as database 132, printer and file resources. In some embodiments of the invention, remote server 130 runs one of the variations of the Microsoft Windows operating systems described above. In other varying embodiments, remote server 130 runs a variant of the UNIX operating system, such as Linux, Solaris, or HP-UX. The embodiments of the invention is not limited to any particular operating system.

FIG. 2 is a diagram of the hardware and operating environment in conjunction

with which embodiments of the invention may be practiced. The description of FIG. 2 is intended to provide a brief, general description of suitable computer hardware and a suitable computing environment in conjunction with which the embodiments of the invention may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a personal computer or a server computer. Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types.

Moreover, those skilled in the art will appreciate that the embodiments of the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. Embodiments of the invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

As shown in FIG. 2, the computing system 200 includes a processor. Embodiments of the invention can be implemented on computers based upon microprocessors such as the PENTIUM® family of microprocessors manufactured by the Intel Corporation, the MIPS® family of microprocessors from the Silicon Graphics Corporation, the POWERPC® family of microprocessors from both the Motorola Corporation and the IBM Corporation, the PRECISION ARCHITECTURE® family of microprocessors from the Hewlett-Packard Company, the SPARC® family of microprocessors from the Sun Microsystems Corporation, or the ALPHA® family of microprocessors from the Compaq Computer Corporation. Computing system 200 represents any personal computer, laptop, server, or even a battery-powered, pocket-sized, mobile computer known as a hand-held PC.

The computing system 200 includes system memory 213 (including read-only memory (ROM) 214 and random access memory (RAM) 215), which is connected to

the processor 212 by a system data/address bus 216. ROM 214 represents any device that is primarily read-only including electrically erasable programmable read-only memory (EEPROM), flash memory, etc. RAM 215 represents any random access memory such as Synchronous Dynamic Random Access Memory.

Within the computing system 200, input/output bus 218 is connected to the data/address bus 216 via bus controller 219. In one embodiment, input/output bus 218 is implemented as a standard Peripheral Component Interconnect (PCI) bus. The bus controller 219 examines all signals from the processor 212 to route the signals to the appropriate bus. Signals between the processor 212 and the system memory 213 are merely passed through the bus controller 219. However, signals from the processor 212 intended for devices other than system memory 213 are routed onto the input/output bus 218.

Various devices are connected to the input/output bus 218 including hard disk drive 220, floppy drive 221 that is used to read floppy disk 251, and optical drive 222, such as a CD-ROM drive that is used to read an optical disk 252. The video display 224 or other kind of display device is connected to the input/output bus 218 via a video adapter 225.

A user enters commands and information into the computing system 200 by using a keyboard 40 and/or pointing device, such as a mouse 42, which are connected to bus 218 via input/output ports 228. Other types of pointing devices (not shown in FIG. 2) include track pads, track balls, joy sticks, data gloves, head trackers, and other devices suitable for positioning a cursor on the video display 224.

As shown in FIG. 2, the computing system 200 also includes a modem 229. Although illustrated in FIG. 2 as external to the computing system 200, those of ordinary skill in the art will quickly recognize that the modem 229 may also be internal to the computing system 200. The modem 229 is typically used to communicate over wide area networks (not shown), such as the global Internet. The computing system may also contain a network interface card 53, as is known in the art, for communication over a network.

Software applications 236 and data are typically stored via one of the memory storage devices, which may include the hard disk 220, floppy disk 251, CD-ROM 252 and are copied to RAM 215 for execution. In one embodiment, however, software applications 236 are stored in ROM 214 and are copied to RAM 215 for execution or are executed directly from ROM 214.

In general, the operating system 235 executes software applications 236 and carries out instructions issued by the user. For example, when the user wants to load a software application 236, the operating system 235 interprets the instruction and causes the processor 212 to load software application 236 into RAM 215 from either the hard disk 220 or the optical disk 252. Once software application 236 is loaded into the RAM 215, it can be used by the processor 212. In case of large software applications 236, processor 212 loads various portions of program modules into RAM 215 as needed.

The Basic Input/Output System (BIOS) 217 for the computing system 200 is stored in ROM 214 and is loaded into RAM 215 upon booting. Those skilled in the art will recognize that the BIOS 217 is a set of basic executable routines that have conventionally helped to transfer information between the computing resources within the computing system 200. These low-level service routines are used by operating system 235 or other software applications 236.

In one embodiment computing system 200 includes a registry (not shown) which is a system database that holds configuration information for computing system 200. For example, Windows® 95, Windows 98®, Windows® NT, and Windows 2000® by Microsoft maintain the registry in two hidden files, called USER.DAT and SYSTEM.DAT, located on a permanent storage device such as an internal disk.

Software Environment

The embodiments of the invention describe a software environment of systems and methods that provide for the automatic update of software components of a distributed system as the various components evolve. The software environment

described in this section operate in distributed systems environments, such as that described above in reference to FIG. 1.

FIG. 3 illustrates an exemplary software environment in which embodiments of the invention operate. The exemplary environment includes an application 308, a network element operating system (OS) 320, and an application bank 330. Network element operating system 320 executes on network element hardware 120 (FIG. 1), and provides the software required implementing network element functions. This software includes configuration and application elements.

Application 308 is a software component that communicates with network element operating system 320. In one embodiment of the invention, application 308 is a management application operative to provide the ability for a remote user on remote systems 102 to control and configure aspects of a network element 120 by communication with network element operating system 320. Application 308 can be a client application, or it can be a server application, the invention is not limited to any particular application type. In some embodiments of the invention, application 308 is divided into three components, a user interface (UI) component 302, an application component 304, and a communication component 306. Dividing the application 308 into these components is desirable, because it provides the ability to change each one of them separately or to load them on different computers. Furthermore, it is desirable to separate the user interface component, because it provides a means for supporting many different types of users.

The UI component 302 provides a mechanism for a user to interface with software controlling a remote component. In one embodiment of the invention, UI component 302 provides a graphical user interface for controlling a network element of a telecommunications system. The user interface provides for the display and input of configuration and control elements used to affect the operation of the network element.

Application component 304 provides element specific functions that interpret the input received from UI 302 and adapt the input for communication to the network element. In some embodiments of the invention, application component 304 comprises

processes and tasks that manage network elements 120. Application component 304 can operate on a demand basis, or it can operate on autonomous and cyclic basis. In some embodiments, application component 304 collects alarm information, performance information, testing results and the like. In alternative embodiments, application component 304 operates as an engine to implement user operations, including complicated user operations.

Communication component 306 implements a communication protocol providing a mechanism for application 308 to communicate with the network elements. The communication protocol can include message sets, including commands and data that manipulate the configuration and operation of the network elements.

The UI component 302, application component 304 and communication component 306 components communicate with one another using a software communications bus 310. In one embodiment of the invention, communication bus 310 uses CORBA (Common Object Request Broker Architecture). In an alternative embodiment of the invention, the communication bus 310 utilizes DCOM (Distributed Component Object Model). In a further alternative embodiment, communication bus 310 uses RMI (Remote Method Invocation). DCOM, CORBA and RMI are all known in the art. In a still further alternative embodiment, communication bus 310 send and receives commands and data using XML (eXtensible Markup Language).

Application 308 can have multiple versions, as can network element operating system 320. Each version of application 308 is associated with a version of network element operating system 320. Moreover, in those embodiments in which application 308 comprises UI component 302, application component 304, and communications component 306, the three individual components can also have one or more versions. The versions of the components are also associated with a version of network element software 320. When a version of an application is "associated" with a version of network element software, the two elements are compatible with one another, and have been tested and verified as working together. A version of an application 308 (and components 302, 304 and 306) may be associated with more than one version of

network element operating system 320. Conversely a version of network element operating system 320 may be associated with more than one application 308 and/or components 302, 304 or 306.

Application bank 330 maintains association data, and operates to store varying versions of application 308, user interface component 302, application component 304 and communications component 306. In one embodiment of the invention, application bank 330 maintains the associations in a database, such as database 332.

FIG. 4 is a flowchart of a method 400 illustrating the operation of the above-described components. When a user at a remote workstation *102* invokes an application 308 to interact with a network element 120 running a version of network element software 320, a further component of application 308, application controller 312, connects with the desired network element address (block 402). The application controller communicates with the desired network element software to determine its version (block 404). A check is then made to determine if the appropriate application 308 and/or application components 302, 304 and 306 that are associated with network element software 320 currently exist on the remote workstation *102 (block 406)*. If the correct associated versions exist, they are then loaded and executed (block 410). Otherwise the application controller 312 communicates with application bank 330, and retrieves the correct version (block 408). After retrieving the correct version of application 308 and/or components 302, 304 and 306, the application is loaded and activated (block 410).

This section has described the various software components in a system that manages the application updates required as component software evolves. As those of skill in the art will appreciate, the software can be written in any of a number of programming languages known in the art, including but not limited to C/C++, Visual Basic, Smalltalk, Pascal, Ada and similar programming languages. The invention is not limited to any particular programming language for implementation.

Conclusion

Systems and methods for supporting evolution of versions in distributed systems are disclosed. The embodiments of the invention provide numerous advantages over previous systems. The system automatically determines applications and application components that are compatible with distributed components, and loads the appropriate versions as necessary. The application and application components are more stable, because there is no need to build into the code version checks and special behavior based on the version checking. The application and application components need only maintain compatibility with a particular version of the distributed component. This reduces the development effort, and also reduces the amount of time required for testing, both of which reduce the overall development cost and time to market. Furthermore, the risk of software bugs existing in the application and application components is reduced, because a particular version will only contain the relevant instructions for that version, and need not check and provide for compatibility with different versions in the field. Fewer conditional statements mean fewer bugs, fewer tests, and less risk.

Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiments shown. For example, the systems and methods have been discussed in the context of a telecommunications system in which management applications communicate with network element software. However, the systems and methods of the invention can be applied to any distributed system. This application is intended to cover any adaptations or variations of the present invention.

The terminology used in this application is meant to include all of these environments. It is to be understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reviewing the above description. Therefore, it is manifestly

intended that this invention be limited only by the following claims and equivalents thereof.

What is claimed is:

1. A computerized system for managing application software versions, the system comprising:

a remote element component having a remote version identification;

a remote client;

an application bank operative to store a set of applications, each of the applications having an application version identification; and

wherein upon a request by the remote client, the application bank is further operative to retrieve at least one of the set of applications having an application version identification that corresponds to the remote element version identification and to transfer the at least one of the set of applications to the remote client.

2. The system of claim 1, wherein the at least one of the set of applications comprises an element management application.

3. The system of claim 2, wherein the element management application further comprises a user interface component having an interface version identification; and

wherein the application bank is further operative to store the user interface component and to transfer the user interface component to the remote client when the interface version identification is associated with the element version identification.

4. The system of claim 2, wherein the element management application further comprises an application component having an application component version identification; and

wherein the application bank is further operative to store the application component and to transfer the application component to the remote client when the application component version identification is associated with the element version identification.

5. The system of claim 2, wherein the element management application further comprises a communication having a communication component version identification; and

wherein the application bank is further operative to store the communication component and to transfer the application component to the remote client when the communication component version identification is associated with the element version identification.

6. The system of claim 1, wherein the remote element is a network element operating system for a network element in a telecommunications system.

7. The system of claim 2, wherein the element management application further includes an application component, a user interface component, and a communications component.

8. A method for managing software compatibility in a multi-component system, the method comprising:

connecting to a first component;

determining a version for the first component;

determining if an application that communicates with the first component has a version corresponding to the version of the first component resident on a client system, and if not, then retrieving the application from an application bank; and

activating the application.

9. The method of claim 8, wherein the first component is a network element in a telecommunications system.

10. The method of claim 8, wherein the application that communicates with the first component is a network element management application.

11. The method of claim 10, wherein determining if an application that communicates with the first component has a version corresponding to the version of the first component is resident on a client system further determines if a user interface component of the application is resident on the client system, and if not, then retrieving the user interface component from the application bank.

12. The method of claim 10, wherein determining if an application that communicates with the first component has a version corresponding to the version of the first component is resident on a client system further determines if an application component of the application is resident on the client system, and if not, then retrieving the application component from the application bank.

13. The method of claim 10, wherein determining if an application that communicates with the first component has a version corresponding to the version of the first component is resident on a client system further determines if a communications component of the application is resident on the client system, and if not, then retrieving the communications component from the application bank.

14. A computer-readable medium having computer executable instructions for performing a method for managing software compatibility in a multi-component system, the method comprising:

connecting to a first component;

determining a version for the first component;

determining if an application that communicates with the first component has a version corresponding to the version of the first component is resident on a client system, and if not, then retrieving the application from an application bank; and

activating the application.

15. The computer-readable medium of claim 14, wherein the first component is a network element in a telecommunications system.

16. The computer-readable medium of claim 14, wherein the application that communicates with the first component is a network element management application.

17. The computer-readable medium of claim 16, wherein determining if an application that communicates with the first component has a version corresponding to the version of the first component is resident on a client system further determines if a user interface component of the application is resident on the client system, and if not, then retrieving the user interface component from the application bank.

18. The computer-readable medium of claim 16, wherein determining if an application that communicates with the first component has a version corresponding to the version of the first component is resident on a client system further determines if an application component of the application is resident on the client system, and if not, then retrieving the application component from the application bank.

19. The computer-readable medium of claim 16, wherein determining if an application that communicates with the first component has a version corresponding to the version of the first component is resident on a client system further determines if a communications component of the application is resident on the client system, and if not, then retrieving the communications component from the application bank.

20. A method for managing software compatibility in a telecommunications network, the method comprising:

connecting to a network element;

determining a version for the network element;

determining if an element management application that communicates with the network element has a version corresponding to the version of the element management application on a client system, and if not, then retrieving the element management application from an application bank; and

activating the element management application.

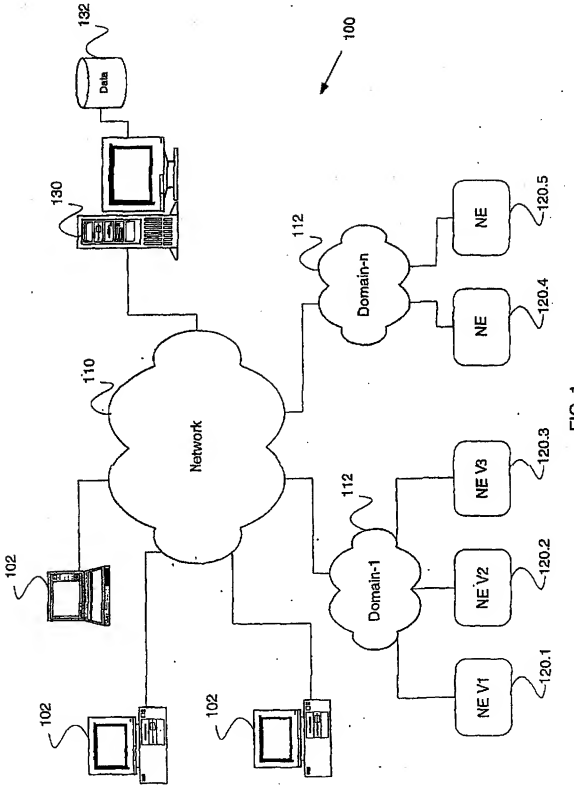

100
132
Data
130
112
Domain-n
NE
120.5
NE
120.4
110
Network
112
Domain-1
NE V3
120.3
NE V2
120.2
NE V1
120.1
102
102
102

FIG. 1

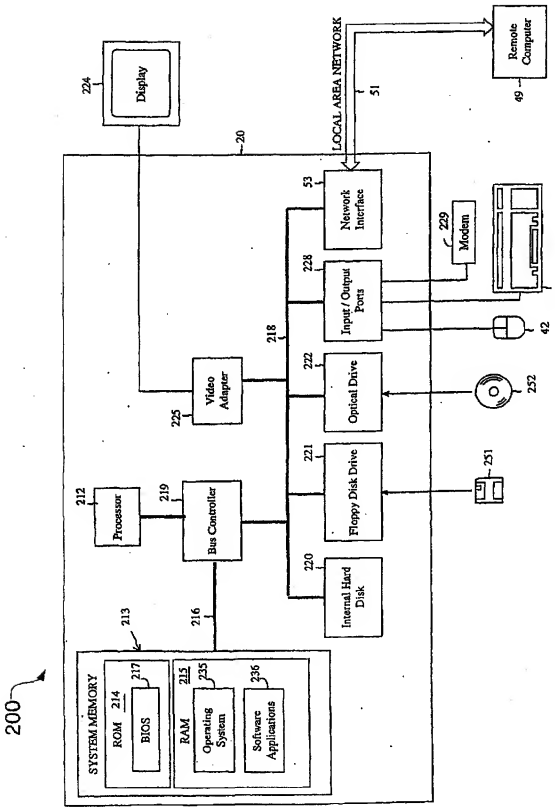

200
SYSTEM MEMORY
ROM 214
BIOS 217
RAM 215
Operating System 235
Software Applications 236
213
216
Processor 212
Bus Controller 219
Internal Hard Disk 220
Floppy Disk Drive 221
Optical Drive 222
Video Adapter 225
Input / Output Ports 228
Network Interface 53
Modem 229
218
20
Display 224
LOCAL AREA NETWORK
51
Remote Computer 49
40
42
252
251

FIG. 2

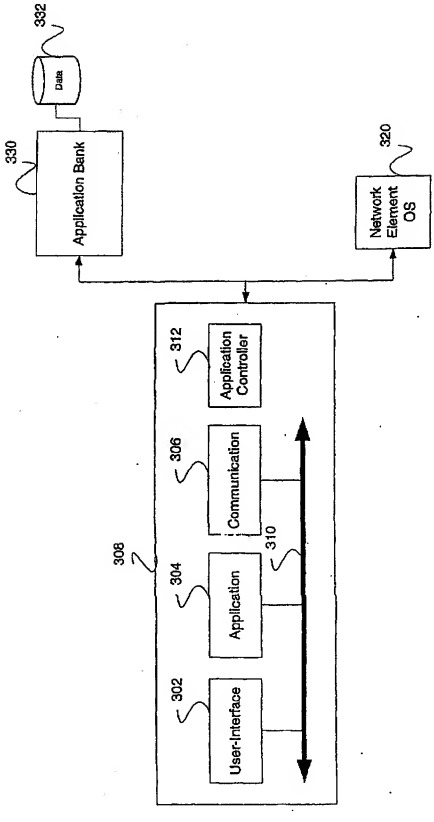

332
Data
330
Application Bank
320
Network Element OS
312
Application Controller
306
Communication
310
308
304
Application
302
User-Interface

FIG. 3

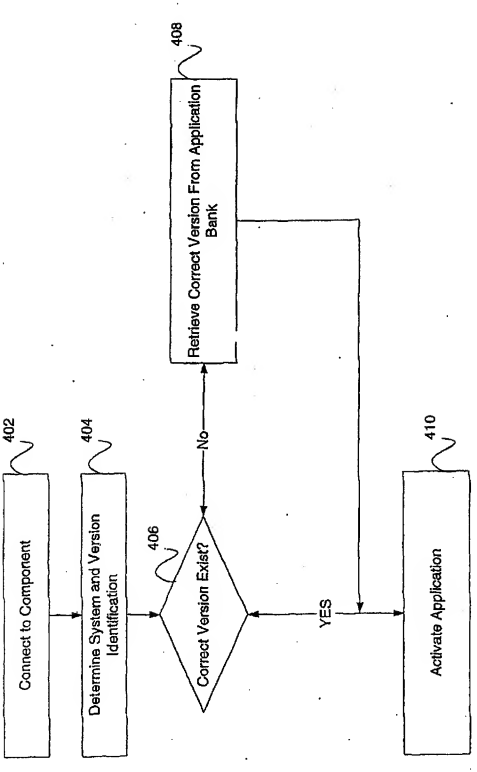

402
Connect to Component
404
Determine System and Version Identification
406
Correct Version Exist?
No
408
Retrieve Correct Version From Application Bank
YES
410
Activate Application

FIG. 4